

6.01 基础设施指南

本指南概述了 6.01 硬件和软件基础设施。我们使用 ActivMedia Pioneer 3DX 机器人，连同一些我们自己发明的硬件，来支持各种机器人感知与控制以及电路设计与构建方面的实验。此外，我们还有一个程序，用于模拟机器人在世界中移动时的感知和动作。

我们的机器人有一个内置计算机，用于进行低级控制和管理传感器；这台计算机通过串行电缆与笔记本电脑进行交互。

我们使用一个名为 *soar* 的 Python 程序来控制真实的或模拟的机器人，该程序在笔记本电脑上运行。*soar* 允许您管理是与真实机器人还是模拟器进行通信，并允许您使用操纵杆或软件“大脑”来驱动。大脑使用传感器模型（如阅读材料第 1 章所述）来控制机器人，这意味着大脑程序的任务是将当前的感觉输入映射到下一个动作；它每秒会执行几次。您可以在自己的计算机或 Athena* 机器上运行 *soar* 模拟器，但要控制机器人，您需要在实验室的笔记本电脑上运行。以下描述适用于实验室的笔记本电脑，但在您的笔记本电脑或 Athena* 上使用 *soar* 模拟器是类似的。

1 机器人传感器和效应器

我们的机器人有几种不同的传感器（获取外部世界信息的方式）和效应器（改变外部世界状态的方式）。

轮子

这些机器人有两个驱动轮，每边一个，后面有一个脚轮。您可以通过指定向前移动的速度（以米/秒为单位）和旋转速度（以弧度/秒为单位）来指挥机器人。控制软件会将这些速度转换为每个轮子的有符号速度。

声纳

机器人有八个超声波换能器（俗称声纳）；这些是前部的金色圆盘。它们发出声脉冲并监听反射声。原始传感器返回飞行时间值，即声音反弹回来的时间。机器人的处理器将这些值转换为距离估计值。

当声纳发出一个声脉冲时，会有一个被称为“衰减期”的阶段，在此期间传感器仍在因产生脉冲而振动，无法有效地监听回声。如果一个物体距离太近，以至于在衰减期内就返回回声，传感器就无法检测到该物体。同样，距离太远的物体也不会返回回声。如果最近的物体距离机器人 10 厘米到 1 米之间，传感器通常相当可靠。如果传感器未检测到回声，则返回的距离会远远超出传感器的范围。

根据声纳传感器前方物体的材质，声脉冲可能会向其他方向反射，而不是反射回传感器。这也导致传感器无法检测到物体。为了解决这个问题，我们覆盖了所有的“墙壁”。

*Athena is MIT's UNIX-based computing environment. OCW does not provide access to it.

用气泡膜，这会导致脉冲向多个方向从墙壁反弹，使得传感器很可能会检测到反射。

里程计

该机器人的每个驱动轮上都有轴编码器，用于计算轮子的旋转次数（分数）。机器人处理器利用这些编码器的计数来更新机器人在全局参考系中的估计位姿（一个同时表示位置和方向的术语）。

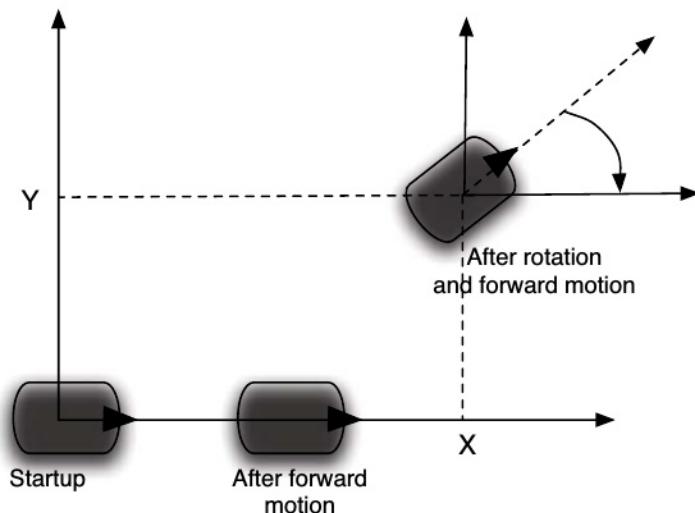


图1 全球里程计参考系

图1展示了机器人里程计报告的参考系。当机器人启动时，该参考系初始化为原点位于机器人中心，正x轴指向机器人前方。现在你可以把那个参考系想象成涂在地面上的；当你移动或旋转机器人时，它会持续报告其在那个原始参考系中的位姿。

请注意，如果您拿起真正的机器人并移动它，而轮子不转动，姿态读取值不会改变。它只知道它移动了，是因为轮子的转动。并且要记住，里程计远非完美：轮子打滑，误差（尤其是旋转误差）会随着时间的推移而累积。

模拟 - 数字接口

我们可以通过模拟数字接口将额外的传感器和效应器连接到机器人上。机器人上有一根电缆，连接到四个模拟数字转换器（这些转换器允许我们将输入电线上的电压转换为可以通过软件读取的值）和一个数字模拟转换器（这允许我们将软件中写入的数字转换为输出电压）。有关这些输入的硬件连接的详细信息，请参阅第7.4节和第7.5节。

2 大脑

一只翱翔的大脑具有以下结构：

```

def setup():
    print "Loading brain"
def brainStart():
    print "Starting"
def step():
    print "Hello robot!"
def brainStop():
    print "Stopping"

```

设置程序会在解释器读取大脑文件时调用一次。大脑启动程序会在您开始运行大脑时调用。步骤程序会每秒调用多次。上述大脑，如果运行，当您加载（或重新加载）文件时，首先会打印“正在加载大脑”，当按下 SOAR 的启动按钮时会打印“启动”，然后在 SOAR 消息窗口中持续打印“Hello robot!”，直到您停止它，此时它会打印“停止”。

我们通常会使用状态机来控制我们的机器人。这里有一个带有状态机控制器的非常简单的“大脑”：

```

import lib601.sm as sm
from soar.io import io

class MySMClass(sm.SM):
    def getNextValues(self, state, inp):
        return (None, io.Action(fvel = 0.1, rvel = 0))

def setup():
    robot.behavior = MySMClass()
    robot.behavior.start(verbose = True)

def step():
    robot.behavior.step(io.SensorInput()).execute()

```

该程序首先加载一些支持文件。然后，我们展示了简单状态机的定义，它总是生成一个动作，将机器人的前向速度设置为0.1米/秒，旋转速度设置为0。我们将任何消耗SensorInput类实例流并产生Action类实例流的状态机称为行为。请注意，SensorInput和Action都是在名为io的soar模块中定义的，因此我们必须在其前面加上模块名。

设置函数实例化 MySMClass () 行为（如果您想使用不同的行为，在此进行实例化）并启动它。启动时，参数 verbose 设置为 True 会导致打印状态机的输入、输出和状态；您可以通过将该参数设置为 False 来抑制打印（或者从调用中删除可选的 verbose 参数）。

然后，最后，在步长函数中，我们调用 io.SensorInput ()，这会创建 io.SensorInput 类的一个实例，其中包含最近可用的传感器数据。该实例作为输入传递给状态机的 step 方法，该方法会调用其 generateOutput 方法，该方法会返回一个

¹ Alternatively, we could have put the line from io import * at the top of the file, which would then have allowed us to say Action instead of io.Action. As a matter of policy, we try not to do this, because in a complicated software system it can become difficult to figure out what module a particular class or procedure is defined in, and sometimes name clashes (where two modules both define getBest, for example) ensue.

io.Action 类的实例。最后，我们要求该操作执行，这实际上会向机器人发送指令。如果这种写法看起来令人困惑，这里有一个替代版本，能更清楚地说明正在发生什么。

```
def step():
    inp = io.SensorInput()
    act = robot.behavior.step(inp)
    act.execute()
\stoptypine

The attributes of an {\tt io.Action} instance are:
\startitemize
\item {\tt fvel}: a single real number specifying forward velocity in
  meters per second. A positive value moves the robot forward, and a
  negative value moves the robot backward.
\item {\tt rvel}: a single real number specifying rotational velocity
  in radians per second. A positive number causes the robot to turn
  left and a negative number causes the robot to turn right.
\item {\tt voltage}: a single number in the range 0 to 10, specifying
  the voltage to be transmitted to the digital-to-analog converter
\stopitemize
Passing those three values into the initializer will make a new
instance. Calling the {\tt execute()} method on an {\tt Action}
instance will cause the robot to execute that action.
```

```
The attributes of a {\tt io.SensorInput} instance are
\startitemize
\item {\tt sonars}: a list of 8 values from the sonar sensors;
  distances measured in meters; with the leftmost sensor being number
  0
\item {\tt odometry}: an instance of the {\tt util.Pose} class, which is
  described in more detail in the online software documentation
\item {\tt analogInputs}: a list of 4 values from the
  analog-to-digital encoders, in the range 0 to 10.
\stopitemize
In the simulator only, it can sometimes be useful to ‘‘cheat’’ and
make reference to the robot’s {\tt actual} coordinates in the global
coordinate frame of the simulator (rather than in the odometry frame
which is where the robot was when it was started). If you make an
instance of {\tt io.SensorInput} this way:
\starttyping
s = io.SensorInput(cheat = True)
```

然后，里程计的值将代表模拟器中的实际坐标。在机器人上执行此操作会导致错误。使用作弊姿态产生真正影响的情况是，当您在屏幕上拖动模拟的机器人时：常规的里程计并不知道机器人已经移动（因为轮子没有转动），但作弊姿态仍然会确切地知道它最终的位置。仅将其用于特殊调试，因为真正的机器人无法进行此类操作！

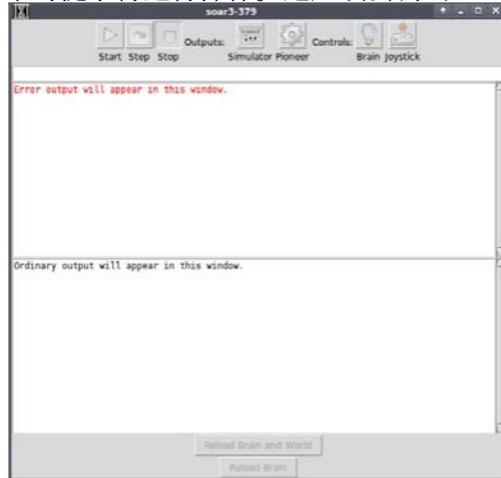
3 使用 Soar 模拟器

若要用大脑控制模拟机器人，请执行以下操作：

· 类型

> 翱翔；高飞

在您计算机的终端窗口中对提示符进行操作。这应该会调出 SOAR 窗口，



它看起来像这样：

- 点击模拟器按钮，并选择 tutorial.py 世界。
- 点击“大脑”按钮，使用文件对话框导航至“桌面/6.01/designLab02/smBrain.py”并选择它。
- 点击“开始”按钮。

机器人会向前行驶，并在窗口中打印一些输出内容。若要停止机器人，请点击“停止”按钮。如果它持续打印了一会儿，别担心。所有那些文本都已存储在某个地方，等待被打印出来。

一般来说，您在一个文件中定义一个 Soar 大脑并将其加载到 Soar 中。如果在加载大脑时出现错误，您将看到一条错误消息（红色显示）。如果您需要编辑大脑文件，您可以在编辑器中进行编辑，然后按“重新加载大脑”按钮告知 Soar 重新加载大脑。“重新加载大脑和世界”按钮也会重新加载大脑，并且在模拟器中，它会将机器人重新放置在初始位置。在先锋机器人上，重新加载世界会重置与机器人的串行连接，这会使机器人将其里程计重置为原点。

对于模拟器上的声纳，你会看到机器人上不同点伸出线条：这些代表声纳传感器的信号路径。这些模拟传感器遵循一个理想模型，即它们总是测量沿该射线到第一个接触到的表面的距离（带有少量添加的随机噪声）。当声纳未检测到任何反射时，线条会以红色显示。在模拟器中，当大脑运行时，你可以用鼠标拿起并拖动机器人，观察读数如何变化。

您可能会注意到模拟的机器人偶尔会变成红色。这意味着它接近或与墙壁发生碰撞。如果它卡住了，而您向它发送向前移动的指令，什么都不会发生。不过，您仍然可以后退或转身。

²If you use idle as the editor, remember that you *don't* run the brain with “run module” from the run menu. You can, however, check the syntax of your code with “check module,” which could be a useful thing to do before loading it into soar.

4 使用先锋机器人

您可以通过以下步骤将机器人连接到实验室笔记本电脑：（1）将一条短的蓝色 USB 适配器电缆连接到您的笔记本电脑上；（2）将 USB 适配器连接到串行电缆的一端；（3）将串行电缆的另一端连接到机器人上的串行连接器上，该连接器位于机器人侧面的带有电源开关和指示灯的面板附近。

在将串行电缆插入之前，一定要在机器人背面的手柄上简单地打个结将其固定住。这样可以缓解拉力，防止串行电缆从机器人上被拉扯下来，或者在电缆被拉扯时弄断连接器。

首先开启机器人。若要使用操纵杆驱动机器人，请点击“**先锋**”和“**操纵杆**”，然后点击“**开始**”。在生成的操纵杆窗口中点击并拖动，即可控制机器人的前进和旋转速度。

提示

用于运行机器人：

- 当你打算运行机器人时，请确保机器人已开启，且串行电缆两端都已连接好。
- 这些机器人相当坚固，但是（a）掉落它们或者（b）让它们撞到墙上可能会损坏它们。请小心。如果机器人开始远离您，**抓住它并捡起来**然后关闭电源。
- 有时，您可以通过将机器人后仰，使其轮子悬空来进行基本的调试，这样您就能看到轮子转动（以及转动的方向），而机器人本身实际上并不会移动。
- 如果机器人看起来完全死机了，也就是说，接通电源时没有指示灯亮起，那可能是保险丝烧毁了。
与工作人员沟通。
- 如果你运气不佳，试试这个过程：
 - 退出翱翔并关闭机器人；
 - 重新启动翱翔并开启机器人；
 - 在那之后，在 SOAR 中选取机器人图标；然后
 - 倾听声纳发出的“滴答滴答”的声音。
- 如果串行电缆由于某种原因未插好，请执行上述步骤。

5 追踪与绘图

Soar 具有进行各种跟踪和绘图的设施，用于调试您的程序和记录您的结果。在此，我们概述了这些设施中的每一项。

5.1 了解机器人所看到的

通常，了解机器人传感器的输出信息非常重要，要么是为了理解机器人为何以这种方式运行，要么是为了记录实验结果。有一些有用的工具可以帮助实现这一点。要使用这些工具中的许多，您必须在初始化函数中创建一个RobotGraphics实例。这通常看起来像这样：

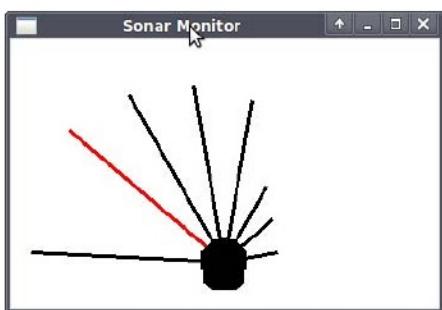
导入图形库

```
def setup():
    robot.gfx = gfx.RobotGraphics(sonarMonitor = True,
                                  drawSlimeTrail = True)
```

当大脑停止运行时，会绘制粘液轨迹和静态图形（如下文所述）。这总是在您按下“**停止**”按钮时进行，但如果您希望在您的控制状态机完成时自动停止大脑，则可以在您的步骤函数中添加以下一行：

```
io.done(机器人行为是否完成)
```

5.1.1 声纳监测仪



实时查看机器人的声纳读数可能会有用，这既可以检测硬件问题，也可以作为调试工具，以更好地理解大脑的行为。在左侧看到的声纳显示器提供了机器人声纳读数的图形显示，无论 SOAR 是连接到模拟器机器人还是先锋机器人。就像模拟器中的声纳一样，声纳由一条黑线表示，当声纳完全听不到回声时，黑线会变成红色。

你可以控制声纳监测仪是否关闭。

通过更改“声纳监视器”参数来播放

RobotGraphics 构造函数，它是一个布尔值。默认值为 False。

5.1.2 粘液痕迹

在执行期间，了解机器人所处的位置通常很有用。在真正的机器人上，我们无法确切知晓这一点，但通过查看里程计报告的位姿，我们可以对正在发生的事情有一些了解。我们通过绘制机器人路径的“粘液轨迹”来实现这一点，就好像它是一只蜗牛。轨迹的颜色沿着轨迹连续变化，让您了解特定位置被占据的时间。图 2 展示了一个示例的粘液轨迹。

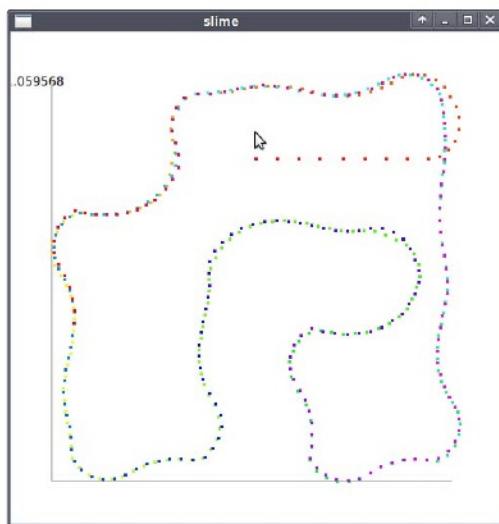


图 2

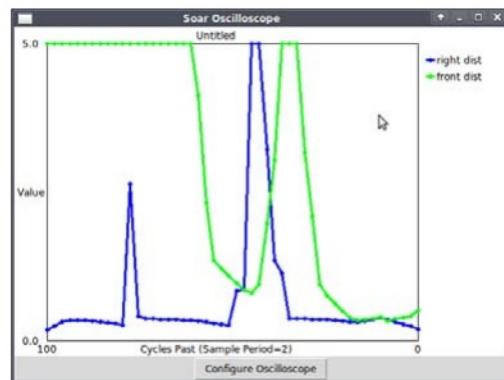
在教程模拟器世界中，一种沿墙行走的机器人的黏液轨迹。

如果将 RobotGraphics 构造函数的 drawSlimeTrail 参数设置为 True，当大脑停止运行时，会弹出一个粘液轨迹窗口。如果您在模拟器上运行，可以将 drawSlimeTrail 设置为 ‘Cheat’，这将使用机器人的真实位置而不是机器人内部的里程计测量值，这样当您拖动机器人时，粘液轨迹将显示其运动。

5.1.3 动态绘图

另一个有用的工具是虚拟示波器，它提供机器人传感器值的某个任意函数的实时图形。`addDynamicPlotFunction` 接受一个（名称，函数）元组，其中名称是显示在图例中的字符串，函数是一个接受 `SensorInput` 实例并返回数字的函数。

右侧的图形展示了示波器窗口绘制的由两个最右侧的声纳传感器报告的最小距离（“right dist”），以及两个前部声纳报告的平均距离（“front dist”）。我们通过向设置函数添加以下代码生成了这个图形。



```
robot.gfx.addDynamicPlotFunction('right dist',
    lambda inp: min(inp.sonars[6:]))
robot.gfx.addDynamicPlotFunction('front dist',
    lambda inp: sum(inp.sonars[3:5])/2.0)
```

“配置示波器”按钮允许您调整图形上的坐标轴范围和标签。默认情况下，y 轴范围是迄今为止看到的 y 值的最大值和最小值。

通过提供一个名称列表和一个返回值列表的函数，也可以在一个调用中绘制多个信号，使用 `addDynamicPlotFunction` 函数。例如，以下代码将绘制前三个模拟输入信号，并为每个信号分配一个名称，该名称对应于其在电路实验室中连接的设备：

```
robot.gfx.addDynamicPlotFunction([('neck', 'left', 'right'),
    lambda inp: inp.analogInputs[0:3]])
```

虽然动态绘图可能是一种非常有用的调试工具，但有时您需要将其关闭，以使大脑获得最佳性能，因为图形任务可能会消耗大量的计算周期。

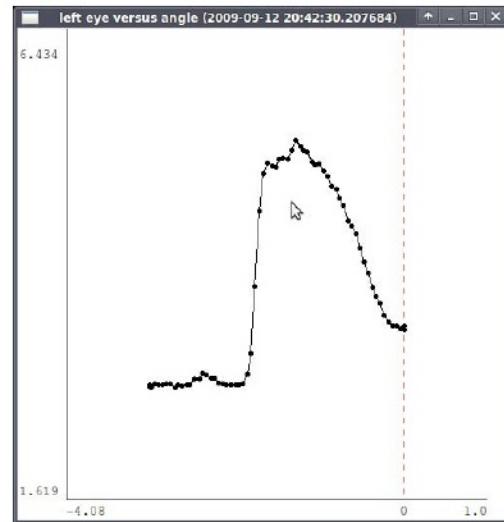
5.1.4 静态绘图

有时，您可能希望将一个信号作为另一个信号的函数来绘制，而不是作为时间的函数来绘制。此外，如果多个信号的幅度差异显著，您可能希望在单独的坐标轴上为它们生成多个信号的绘图。或者您可能会发现实时绘图在每一步上花费太多时间，并导致您的大脑表现异常。在这些情况下，您可以使用静态绘图，它只是存储您想要绘制的值，然后在大脑停止时为每个函数生成绘图窗口。

RobotGraphics 类提供了一个 addStatic - PlotFunction，它类似于 addDynamic - PlotFunction，只不过它可以接收两种不同的信号，因此指定哪一个是 x 轴是很重要的。

信号以及哪个是 y 轴信号。如果未提供 x 参数，则默认将信号作为步长的函数进行绘图。

下面的代码位于设置函数中，它将生成一个类似于图中的图形：机器人左眼的光强度作为机器人相对于初始方向的角度的函数。connectPoints 参数是一个布尔值，用于指定是否在图形中的每个点之间绘制一条线。



```
startTheta = io.SensorInput().odometry.theta
robot.gfx.addStaticPlotFunction(
    x=('angle', lambda inp: startTheta-inp.odometry.theta),
    y=('left eye', lambda inp: inp.analogInputs[1]),
    connectPoints=True)
```

5.2 状态机信息

调试程序时的另一个重要信息来源是程序自身的内部状态。由于我们所有的机器人控制器都是由状态机实例组成的，状态机类提供了许多打印和绘制重要值的技术。

5.2.1 绘制状态机变量

我们可以用与生成传感器值图形大致相同的方式生成状态机输入、状态和输出的静态或动态图形。我们将首先通过一个示例来说明这个过程，然后详细解释步骤。该示例展示了如何生成一个跟随墙壁状态机输出的前向速度的静态图形。

1. 确保感兴趣的州机有一个名称（更多的命名技巧如下）：

```
myMachine = WallFollowerSM()
myMachine.name = 'wallFollow'
def setup():
    robot.behavior = myMachine # could be combination of myMachine with others
```

2. 向 RobotGraphics 实例添加一个静态状态机“探测”：

```
robot.gfx.addStaticPlotSMPProbe(y=('forward vel', 'wallFollow', 'output',
                                lambda outp: outp.fvel))
```

3. 用一组图形任务启动状态机（这通常会默认完成）：

```
robot.behavior.start(robot gfx.tasks())
```

一旦您完成了这三个步骤，当您运行大脑时，您应该会得到状态机数量的静态或动态图。第一步要求您指定要监控的机器。有几种方法可以做到这一点：

- 创建一个状态机的实例，然后添加一个名称属性。

```
myMachine = Increment()
myMachine.name = 'myFavoriteMachine'
```

- 使用其中一个 SM 组合子制造一台机器，并在初始化时提供一个名称：

```
myMachine = sm.Parallel(Increment(), Increment(), name = 'incrementInParallel')
```

- 定义您自己的新的 SM 子类，该子类在初始化器中接受一个名称参数，并设置对象的名称属性：

```
class GoForwardAtSpeed(sm.SM):
    def __init__(self, name = None):
        self.name = name
    def getNextValues(self, state, inp):
        return (None, io.Action(fvel = inp, rvel = 0))
myMachine = GoForwardAtSpeed(name = 'speedy')
```

第二步涉及制作一个状态机探测器并将其提供给 RobotGraphics 实例。状态机探测器是一个元组：（信号名称、机器名称、模式、绘图函数）。

- signalName 是在图表中与该信号相关联的名称。
- machineName 是您想要探测的机器的名称。
- mode 为“输入”“状态”“输出”之一，表明您对机器的这些数量中的哪一个感兴趣。
- plotFunction 是机器的输入、状态或输出的一个函数，用于生成要绘制的数值。

您可以根据具体情况将此类探测器传递给 addStaticSMPProbe 或 addDynamicSMPProbe。

5.2.2 兀长

有时，您感兴趣的值通过打印而非绘图来观察会更好。打印大量调试信息的最简单方法是在调用行为上的 start 操作时，将 verbose 参数设置为 True（默认值为 False），将 compact 参数设置为 False（默认值为 True）。这将在每一步打印每个基本机器的输入、输出和状态，以及组合机器的状态。还会打印出每个机器的名称。默认情况下，名称由系统分配，作为机器实例的类，并加上一个数字以确保唯一性。打印输出看起来会是这样：

```

Step: 0
Cascade_1
    upDownSM Counter = 0
    Sequence_2 Counter = 0
    CountUntil_3 In: Sonar: [5, 5, 1.491, 0.627, 0.330, 0.230, 0.200, 0.265]; Odo:
pose:(-1.633, -1.341, 4.067); Analog: (0.000, 0.000, 0.000, 0.000) Out: -0.400 Next State:
-0.400
Output of upDownSM = -0.4
    forwardSpeedSM In: -0.400 Out: Act: [-0.400, 0, 5.000] Next State: Act: [-0.400, 0,
5.000]
Step: 1
Cascade_1
    upDownSM Counter = 0
    Sequence_2 Counter = 0
    CountUntil_3 In: Sonar: [5, 5, 5, 0.669, 0.356, 0.251, 0.219, 0.289]; Odo:
pose:(-1.614, -1.316, 4.067); Analog: (0.000, 0.000, 0.000, 0.000) Out: -0.300 Next State:
-0.300
Output of upDownSM = -0.3
    forwardSpeedSM In: -0.300 Out: Act: [-0.300, 0, 5.000] Next State: Act: [-0.300, 0,
5.000]

```

如果这种模式过于冗长，您可以做两件事：

1. 在调用 start 时将 compact 参数设置为 True（默认情况下为 True）。在这种情况下，输出看起来会更像这样。您可以看到机器的整个状态，但很难分辨哪些部分属于哪个组成状态机：

```

In: Sonar: [2.465, 1.352, 1.350, 2.465, 1.427, 1.139, 1.060, 0.785]; Odo: pose:(-0.987,
1.493, 2.443); Analog: (0.000, 0.000, 0.000, 0.000) Out: Act: [-0.400, 0, 5.000] Next
State: ((0, (0, -0.5)), None)
In: Sonar: [2.465, 1.381, 1.386, 2.465, 1.466, 1.165, 1.087, 0.739]; Odo: pose:(-0.957,
1.468, 2.443); Analog: (0.000, 0.000, 0.000, 0.000) Out: Act: [-0.300, 0, 5.000] Next
State: ((0, (0, -0.40000000000000002)), None)

```

2. 调用 start 时将 printInput 参数设置为 False。这将跳过输入的打印，如果传感数据妨碍了操作，有时这会很有用。在这种情况下，输出看起来会更像这样：

```

Out: Act: [-0.400, 0, 5.000] Next State: ((0, (0, -0.5)), None)
Out: Act: [-0.300, 0, 5.000] Next State: ((0, (0, -0.40000000000000002)), None)

```

5.2.3 打印状态机值

如果所有这些打印内容过多，而您只想查看特定的值，您可以创建自己的跟踪任务，并在状态机启动时将其提供给它。这些任务只是简单地添加到由 RobotGraphics 实例创建的任务列表中。此示例展示了如何打印出作为主行为机器一部分的两个状态机的状态，这两个状态机已被命名为“sm1”和“sm2”：

```

robot.behavior.start(traceTasks = robot.gfx.tasks() +
    [('sm1', 'state', util.prettyPrint),
     ('sm2', 'state', util.prettyPrint)])

```

状态机跟踪任务非常类似于 5.2.1 节中描述的状态机探测。它是一个元组：（机器名称、模式、跟踪函数）。

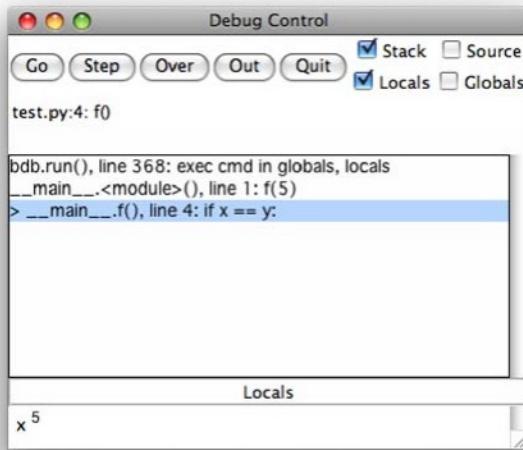
- machineName 是您想要追踪的机器的名称。
- mode 为“输入”“状态”“输出”之一，表明您对机器的这些数量中的哪一个感兴趣。
- traceFunction 是机器的输入、状态或输出的一个函数，它会打印出您希望打印的内容。函数 util.prettyPrint 是我们提供的实用函数，它会以美观、可读的格式打印出很多东西。如果 prettyPrint 不能满足您的需求，欢迎在此使用您自己的函数。

6 在 Idle 中进行调试

Idle 内置了一个调试器。调试有两种方式：(a) 逐步执行您的程序，(b) 等待错误并查看所发生的情况。

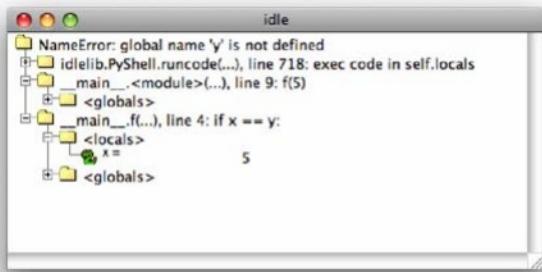
6.1 逐步执行您的程序

点击调试菜单并选择调试器。您将在 Python 外壳中看到[调试开启]，并且会弹出一个窗口。现在，在 Python 外壳中输入您想要计算的表达式。您会看到它在窗口中出现。尝试反复点击“下一步”以查看程序的执行情况。点击“退出”以完成执行，不再进行调试。点击“退出”以退出调试器。



6.2 事后分析

如果在正常执行期间代码出现错误，请点击调试菜单并选择堆栈查看器。这将打开一个窗口，显示当前活动过程的每一行。如果您点击每行左侧的加号，您可以看到全局变量和局部变量的条目，这些是在每次过程调用时活跃的变量绑定。在阅读第 3 章之后，这应该对您来说是有意义的。



7 电路实验室的设备

本节提供了一份实验室中可用的部件清单。

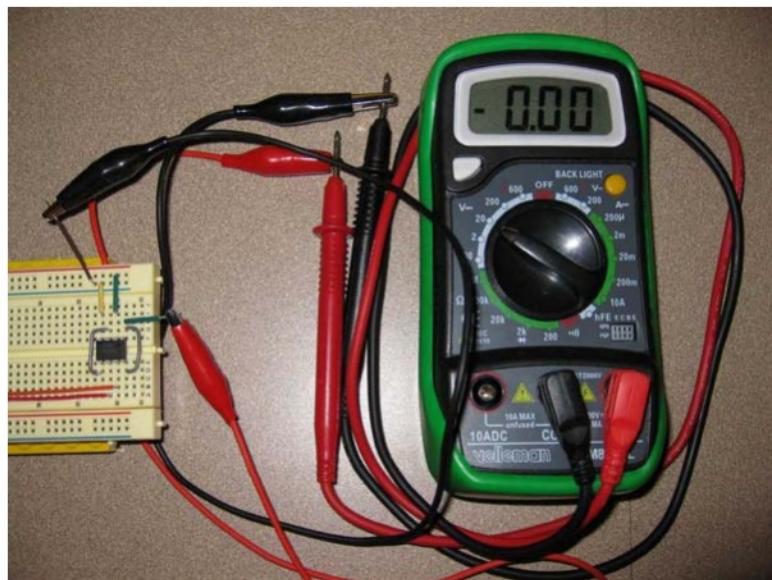
7.1 电源供应

电源有四个端子（一个黑色，三个红色）。要将电线连接到端子，使用两端带有鳄鱼夹的电线。这样就无需拧下端子旋钮。

这是一个电压源。我们将把黑色端子用作负（“接地”）端子，把标有 +15V 的红色端子用作正端子。+15V 端子下方的旋钮实际上控制着这个电源的电压，范围从 0V 到 +15V。

7.2 万用表

我们的万用表允许您测量电流、电压和电阻。您可以使用两根导线将万用表连接到电路。黑色导线应插入接地（公共）插孔。红色导线应插入标有“V - Ω - mA”的插孔，除非测量电流大于 200 mA，此时应使用“10 ADC”插孔。

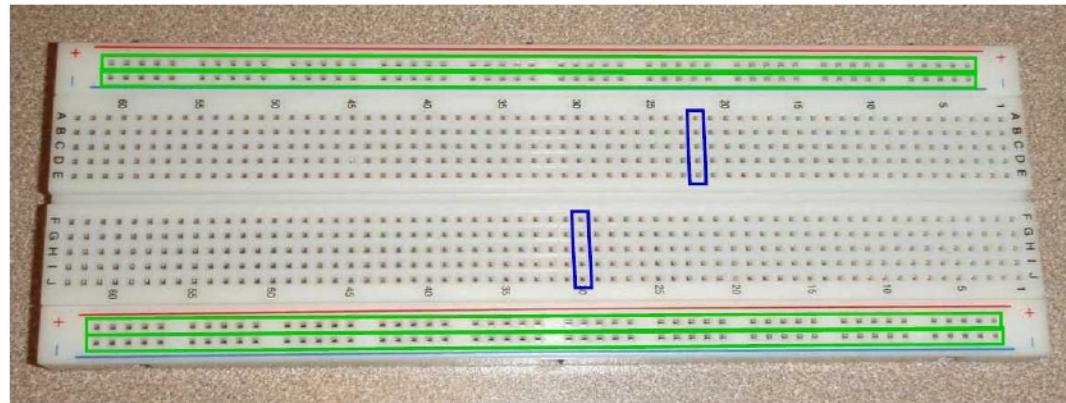


由于计量器的探头较大，它们可能会桥接，从而在小型元件的相邻引脚之间形成不必要的电气连接（“短路”）。这种短路可能会

损坏电路。为避免这种情况，您可以使用如上所示的短导线，通过鳄鱼夹电缆将其连接到仪表探头，来测量电路板各点之间的电阻或电压。鳄鱼夹电缆是两端带有“鳄鱼夹”的彩色屏蔽电缆。您可以挤压鳄鱼夹夹住另一根导线、仪表探头或电源的端子。

7.3 试验电路板

您将在类似于如下所示的“试验电路板”上构建您的电路。



这些试验板上有孔洞，可以插入电线和元件。位于顶部长行的孔洞（标有+）是内部连接的，第二行、底部行和倒数第二行也是如此，如上面的水平（绿色）方框所示。这些行便于分配电源（+10 伏）和接地。中心区域每列 5 个孔洞是内部连接的，如上面两个具有代表性的垂直（蓝色）方框所示。请注意，这 5 列孔洞并非跨接在中央分隔线上。

7.4 与电机的连接

注意：连接到电机和机器人的连接器是不同的。电机连接器有 6 个引脚，机器人连接器有 8 个。

与电机的电气连接由一根带有 RJ11 6P6C 连接器的 6 针电缆提供，该电缆可以通过一个适配器与电路板连接，如下所示。



我们只使用这个连接器的 5 和 6 引脚。将它们分别连接到 +12V 和 GND 会使电机旋转。

7.5 与机器人的连接

注意:机器人与头部的连接器是相同的连接器,所以在插入电缆时,您需要小心不要混淆它们。与电机和与机器人的连接器是不同的。电机连接器有6个引脚,机器人连接器有8个。

机器人与电源的电气连接是通过一根带有 RJ45 8P8C 连接器的 8 针电缆提供的, 该电缆可以通过一个适配器与电路板连接, 如下所示。



引脚 1:	第一人称单数输入 #1
引脚 2:	+10 伏 功率 (限于 0.5 安)
引脚 3:	维2 模拟输入 #2
引脚 4:	地面
引脚 5:	维3 模拟输入 #3
引脚 6:	沃 模拟输出
引脚 7:	维4 模拟输入 #4

这些引脚提供电源以及四个输入通道和一个输出通道。通常情况下, 人们会将引脚 2 (+10 伏) 连接到试验电路板的正电源轨, 将引脚 4 (接地) 连接到负电源 (接地) 轨。

模拟输入必须在 0 到 +10 伏之间, 并以 12 位分辨率转换为数字形式。在 SOAR 中, 可以通过查看 `io.SensorInput` 实例的 `analogInputs` 属性来访问模拟输入, 这是一个形如 `[Vi1, Vi2, Vi3, Vi4]` 的列表。在 SOAR 大脑中, 模拟输入每一步仅更新一次。

机器人产生的模拟输出信号 (`Vo`) 在 0 到 +10 伏之间, 分辨率为 8 位。它可以通过调用 `analogOutput` 来设置, 该函数以 0 到 10 之间的数字作为输入, 并返回 `None`。在 SOAR 大脑中, 模拟输出仅在每一步中设置一次, 使用最近执行的 `io.Action` 实例所设置的值。

7.6 与总部的连接

注意:机器人与头部的连接器是相同的连接器,所以在插入电缆时,您需要小心不要混淆它们。与电机和与机器人的连接器是不同的。电机连接器有6个引脚,机器人连接器有8个。

这里展示了头部连接器的引脚排列。



别针 1:	颈部罐 (顶部)
别针 2:	颈部罐 (中心)
别针 3:	颈壶 (底部)
别针 4:	光敏电阻 (左)
别针 5:	光敏电阻 (常见的)
别针 6:	光敏电阻 (右)
别针 7:	电 荷 (V) M^+ 负电荷 (VM^-)
引脚 18:	电机驱动 + 电机驱动 -

前三个引脚用于电位器，电位器在**7.11节**中有描述。左、右两个光敏电阻的一端都连接到引脚5，另一端分别连接到引脚4（左眼）和引脚6（右眼）。光敏电阻在**7.12节**中有描述。

7.7 电线

我们有很多电线套件，里面装着不同长度的电线，这些电线都是预先裁剪和剥线的。如果可以的话，使用这些电线。尽量选择长度合适的电线，这样它们才能平躺在电路板上。一堆杂乱的电线更难调试，也更容易脱落。请注意，CMax 中的电线颜色与相同长度的实际电线的颜色相符，这应该有助于您巧妙地选择电线。如果您需要更长的电线，可以从线轴上剪下所需长度的电线。使用预先剥线的电线来指导剥线的长度：太少，就无法深入电路板；太多，则会有很多裸露的电线，有与其他电线和元件短路的风险。

7.8 电阻器

我们使用四分之一瓦特电阻器，这意味着在正常情况下它们能够耗散多达 250 毫瓦的热量。超过 250 毫瓦的耗散会导致电阻器过热并自我损坏。



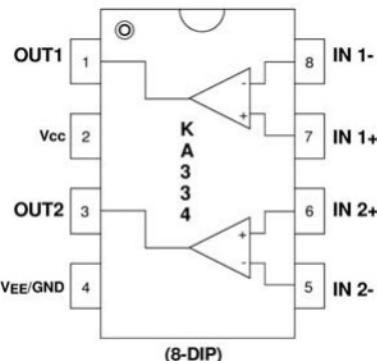
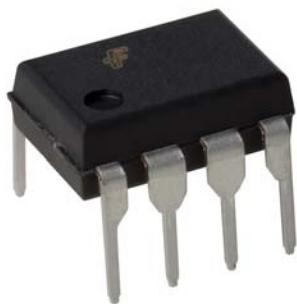
0:	黑色	5:	绿色
1:	棕色 (的)	6:	蓝色
2:	红色	7:	紫罗兰
3:	橙色	8:	灰色
4:	黄色	9:	白色

电阻的值由色带表示。前两个色带表示电阻值的两个有效数字。第三个色带表示10的幂次方。因此，上述电阻器的色带（红色、黄色、绿色）表示电阻值为 24×10^{-5} 欧姆。第四个色带表示电阻值的容差（精度），金色表示 $\pm 5\%$ 。

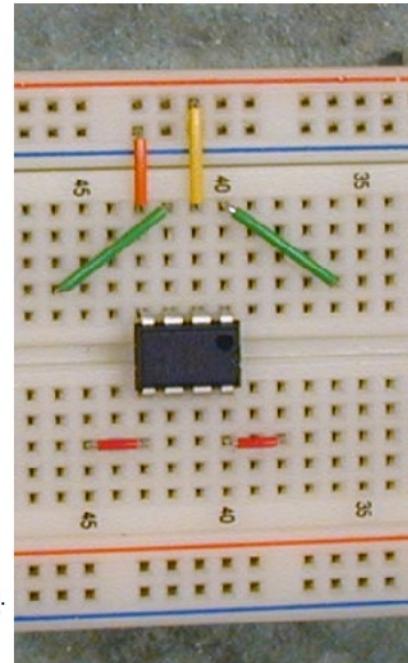
如果您在读取颜色方面有困难，您总是可以用万用表测量电阻。

7.9 运算放大器

我们使用运算放大器（KA334），其封装方式使得两个运算放大器能够适配一个 8 针（双列直插）封装，如下所示。



KA334 internal block diagram
(middle image) © Fairchild Semiconductor. All rights reserved.
This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/fairuse>.



引脚之间的间距使得该封装能够方便地插入如上所示的试验电路板中（请注意，一个点标记为引脚1）。上图顶部的中心（黄色）电线显示运算放大器与正电源（+10伏）的连接。中心左侧较短的（橙色）电线显示与接地端的连接。对角（绿色）电线表示与两个放大器的输出端的连接，而较短的水平（红色）电线表示与两个反相（-）输入端的连接。

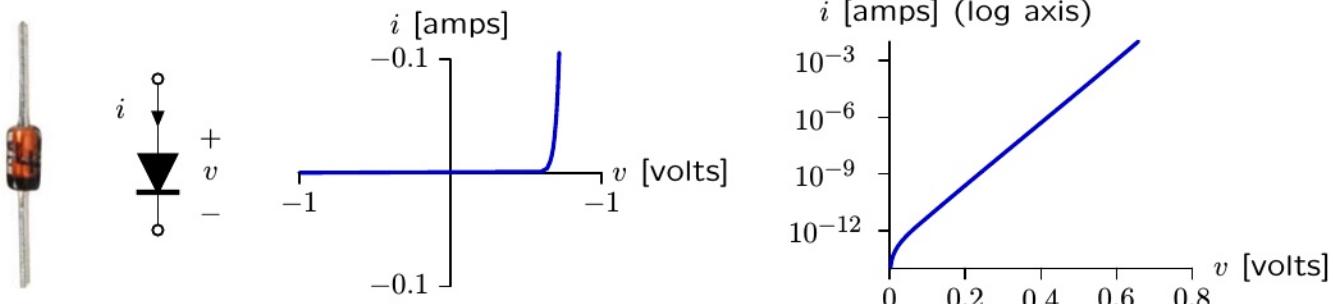
KA334运算放大器能够驱动高达700毫安的输出电流。然而，一个封装中的两个运算放大器的总功耗应限制在1瓦。超过电流或功率限制将损坏运算放大器。

7.10 二极管

二极管是非线性器件：通过二极管的电流随电压呈指数增长。

$$i = I_0 \left(e^{\frac{v}{v_T}} - 1 \right)$$

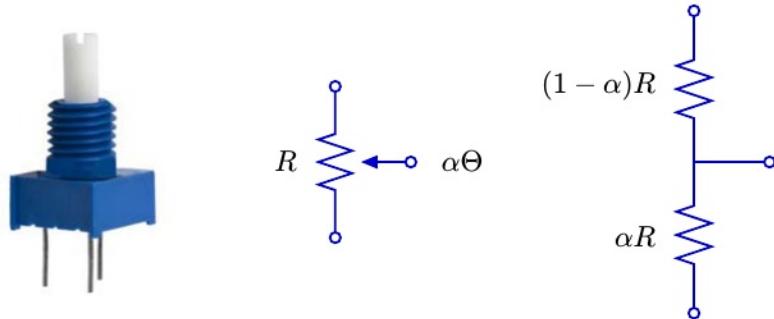
其中， vT 约为 26 毫伏， I_0 约为 0.1 皮安，这是我们在实验室使用的二极管（1N914）的情况。黑色柱条对应二极管的负极。



这些图表在线性坐标轴和对数坐标轴上展示了电压和电流之间的指数关系。

7.11 电位器

电位器（或电位器）是一种三端设备，其电性能取决于其机械轴的角度。下图展示了我们将用于实验的电位器的图片（左）、电位器的电气符号（中）以及等效电路（右）。



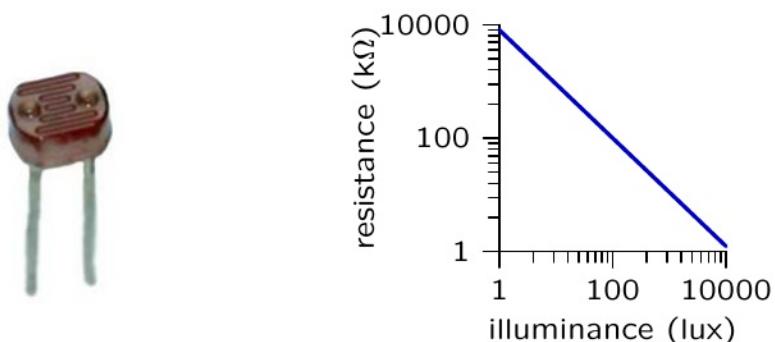
底端和中间端之间的电阻随输入轴的角度 (θ) 成正比增加，而中间端和顶端的电阻则随角度减小，因此顶端和底端电阻的总和保持恒定。我们定义一个比例常数 α ，它随着电位器轴从 0 转到最大角度 Θ （对于我们实验室中使用的电位器，约为 270° ）而介于 0 到 1 之间变化。

通过将一个电位器作为可变电阻连接起来（使用顶部和中间端子），这些端子之间的电阻与轴的角度成正比。

通过将一个电位器作为分压器连接起来（顶部端子连接到电压源，底部端子连接到接地），中间端子的电压就与轴的角度成比例。

7.12 光敏电阻

光敏电阻是一种双端器件，其电阻取决于照射在其表面的光线的强度。光敏电阻由高电阻材料（如硫化镉）制成。入射的光子激发电子——将它们从通常被紧紧束缚的原子中解放出来——这样电子就可以在材料中自由移动，从而传导电流。这种净效应可以通过绘制电阻随入射光强度变化的关系来表征，如下图所示（请注意坐标轴是按对数比例缩放的）。



正常房间的照明强度在 10 到 100 勒克斯之间。一个 60 瓦灯泡（正如我们在实验室中使用的）附近的照度可以超过 10,000 勒克斯。

7.13 光电晶体管

光电晶体管是一种双端器件，其电流取决于照射在其表面的光的强度。光电晶体管是一种半导体器件，入射的光子在其内部产生空穴和电子。我们的光电晶体管工作时，较短的引线（“集电极”）比较长的引线（“发射极”）的正电位更高，这样光生电流就会通过一个集成晶体管（NPN型）进行放大。



覆盖光电管的塑料透镜从大约30弧度的有限锥面收集光线。对于非常明亮的光源，光电流在大约20毫安时达到峰值。光电管应与电阻串联，并跨接在恒定电压电源上，电阻的选择应确保光电管中耗散的热量不超过0.1瓦特。

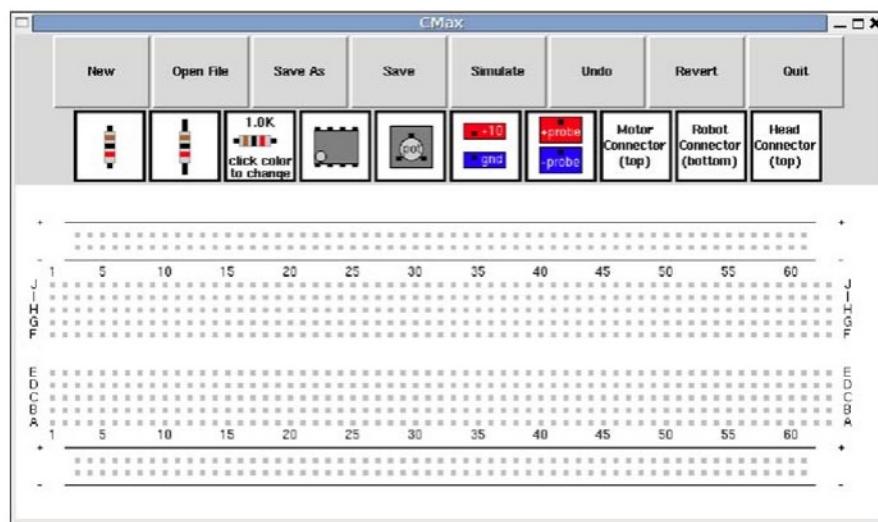
8 最大 C 值

我们将使用一个简单的布局和仿真工具，名为 *Circuits Maximus*，简称“CMax”，来设计并测试电路，然后再进行构建。您可以通过导航到包含 CMax.py 文件的目录并输入以下内容来运行 CMax：

> 运行 CMax.py 文件

8.1 创建和模拟布局

当您启动 CMax 时，您会看到一个包含一排按钮和一个如下所示的电路板图像的窗口。您可以在电路板上添加组件和电线，并模拟所得电路。



添加组件和连线

- 您可以通过点击相应的按钮将组件（电阻器、运算放大器等）放置在电路板上。它们将出现在电路板的左下角，然后您可以将其拖到您希望它们出现在电路上的位置。请注意，CMax 允许您将组件放置在电路板上没有孔洞的位置；这是为了给您留出操作的空间，但要小心不要使任何组件断开连接。
- 若要获取不同方向的组件，请在从菜单中选择它时按住 **Shift** 键。
- 电阻器是带有三条色带的矩形。您可以通过点击原型电阻器图标（带有文字的那个）的色带来更改下一个放置的电阻器的值；它会循环显示颜色。颜色的含义在第 7.8 节中有说明。按住 **Shift** 键并点击电阻器的色带会在另一个方向上循环显示数值。
- 您可以通过点击第一个点并拖到第二个点，用一根电线连接电路板上的任意两个位置。
- 电线端部和元件引脚必须位于代表孔洞的灰色圆点之一上。一个孔洞只能容纳一根电线或一个元件引脚。

- 您必须通过向电路板添加 +10 和接地元件将其连接到电源和接地。每种元件必须恰好有一个。
- 您可以使用**电机连接器**上的引脚 5 和 6 按照第 7.4 节所述为电机添加连接。
- 头部连接器通过引脚 7 和 8 连接到电机上。同样位于头部连接器上的颈部电位器位于引脚 1、2 和 3 上（引脚 2 为电位器的触头）。
- 机器人连接器提供第 7.5 节所述的连接。如果您使用机器人连接器来供电，请勿使用单独的电源和接地连接。

修改您的电路

- 您可以按住控制键（您会看到一个骷髅/交叉骨光标），然后点击组件的主体或连线来删除一个组件或连线。
- 您可以通过点击并拖动来移动导线的端点；您可以通过拖动中间部分来移动整根导线。
- 组件的移动和组件的创建可以使用“撤销”来撤销。撤销操作仅有一个级别深度，所以再次点击“撤销”会将该操作重新执行。
- 读取您布局中电阻的值（万一您忘记了色带的意义）

按住 Shift 键并点击电阻器。该电阻器的值将显示在原型电阻按钮中。**文件管理**

- “退出”“保存”“另存为”“新建”和“打开文件”命令应能按您的预期执行。请确保您为保存电路而创建的文件具有.txt 扩展名。“还原”按钮
这将抹去您自上次保存文件以来所做的更改。

运行测试

有几种方法可以查看运行电路时会发生什么。“**模拟**”按钮会运行您的电路；它需要使用一个输入文件，该文件指定了输入到电路中电位器的输入的时间序列。您永远不需要编写输入文件；我们将为您指定它们，以运行特定的测试。

当您首次点击“**模拟**”时，您需要选择一个测试文件。此后它将使用相同的测试文件。如果您按住**Shift 键并点击“模拟”**，它将重新提示您选择一个测试文件，这样您就可以选择一个不同的文件。

- 您可以通过在电路中放置一个+探头和一个-探头（每种探头恰好一个），点击“**模拟**”按钮，并选择文件noInput.py来测量电路中两点之间的电压；它会在您启动Python的窗口打印出探头位置的电压。如果您的电路中存在具有时动态的组件（如电位器或电机），则在模拟时，它还会弹出一个窗口，显示在探头上生成的信号。
- 如果您的电路中有电机，当您点击“**模拟**”时，会弹出一个窗口，显示电机速度的时间序列，单位为弧度/秒。
- 如果您想了解您的电路如何作为输入信号的函数而运行，您可以添加一个电位器。如果您的电路中有电位器，当您点击“**模拟**”时，会弹出一个窗口，显示电位器a值的随时间变化序列，这样您就可以了解您的电路对输入信号的反应是什么。

8.2 键盘快捷键

有几种键盘快捷键：

- “n” : 清除按钮
- “o” : 打开文件按钮
- “p” : 还原文件按钮
- “q” : 退出按钮
- “r” : 模拟（运行）按钮
- “s” : 保存按钮
- “u” : 撤销按钮

8.3 调试

以下是一些常见的问题：

- 未能解出方程！检查是否有短路或冗余电线。例如，这可能是由于将电源连接到接地造成的。
 检查您的布线。也许您无意中将同一列孔用于两种用途。最坏的情况是，您可以系统地移除电线，直到问题消失，这将告诉您问题是什么。
- 元素[‘电线’ , ‘b47’ , ‘b41’]在节点b41处未连接到任何东西。‘b41’ 表示第41列底部的一组五个孔。如果您收到这样的消息，请检查该元素应该连接到什么。您知道，在这种情况下，应该将其他东西插入到第41列的底部部分。
- 非法引脚意味着您有一根电线或一个元件，其端部或引脚位于电路板上没有孔的位置。

麻省理工学院开放式课
程网站 <http://ocw.mit.edu>

6.01SC 电气工程与计算机科学导论 2011 年秋季

有关引用这些材料或我们的使用条款的信息，请访问：<http://ocw.mit.edu/terms>。